MWSUG 2016 – TT13

An Animated Guide: The Internals of PROC REPORT

Russ Lavery, Bryn Mawr PA

ABSTRACT

PROC REPORT is an exciting “big file technique” that every programmer should know. PROC REPORT allows the creation of complicated reports, with many levels of summarization, while only reading a large source file one time. PROC REPORT allows -in just one step- summarization to a desired level, calculation of new variables and the appending of different kinds reports into one complex report. ALL this happens in one read of the input data. Especially interesting is that the internal file for PROC REPORT, a file that holds “the combined multiple report” can be sent to a SAS ® data set and used as a input data.

This paper will attempt to show the time sequence of the internal actions of PROC REPORT. Knowing the time sequence of actions, especially calculations, is crucial to doing complicated PROC REPORTs. This paper supports a highly animated PowerPoint deck where the time sequence could be demonstrated as a sequence of events. This paper must use screen prints from the presentation and words, knowing words are less effective than visuals, to explain the time sequence.

INTRODUCTION

Figure 1 is just an illustration of what is meant by combining multiple reports into one – and doing that in only one pass through the data. Remember a data read is an expensive operation.

I think of Figure 1 as combining two different kinds of reports. Inside the red box I “see” something like a PROC FREQ – a crossing of variable values. This is powerful because PROC REPORT will add columns depending on the number of zones that happen to be in the data set.

Appended to the right we see a different type of report. Three columns show averages for three different variables but there is no “crossing” of the variables involved. An interesting trick is that the internal file for this report can be sent to a SAS table and used in some other way.
The fact that PROC REPORT can produce all of this in one read of the data is why I call PROC REPORT a “big file technique”.

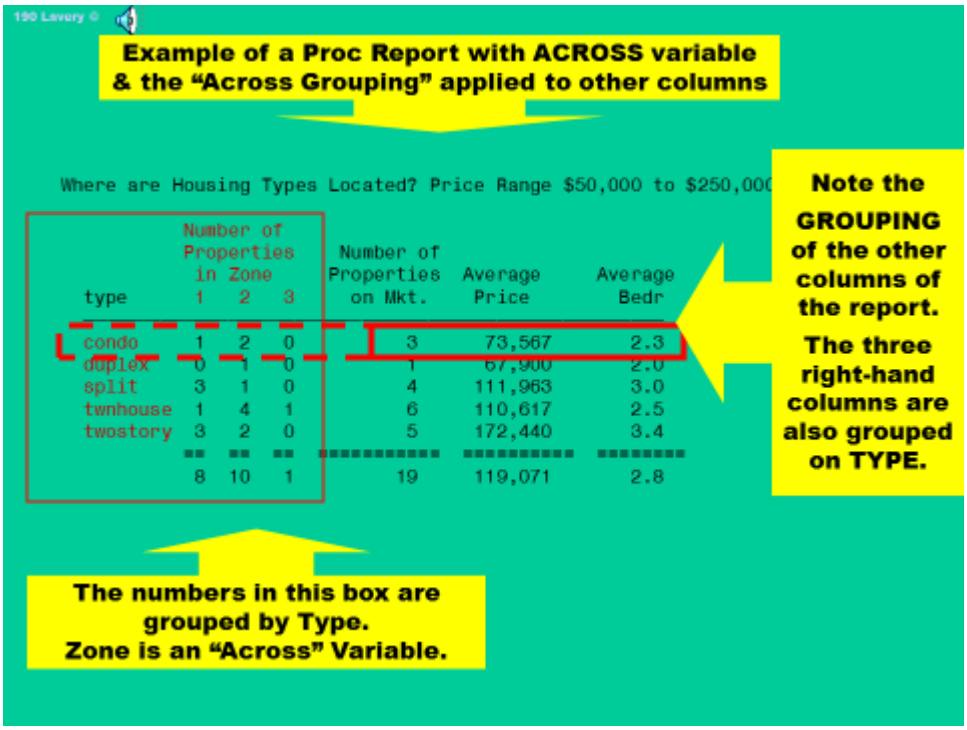

Figure 1

PROC REPORT INTERNALS

Please look at the important graphic in figure 2.
In the upper upper-left-hand corner we see the data set that will be used for our examples – though many examples will have a where clause to limit the data used in the report.

On the left-hand side we see the SAS syntax.

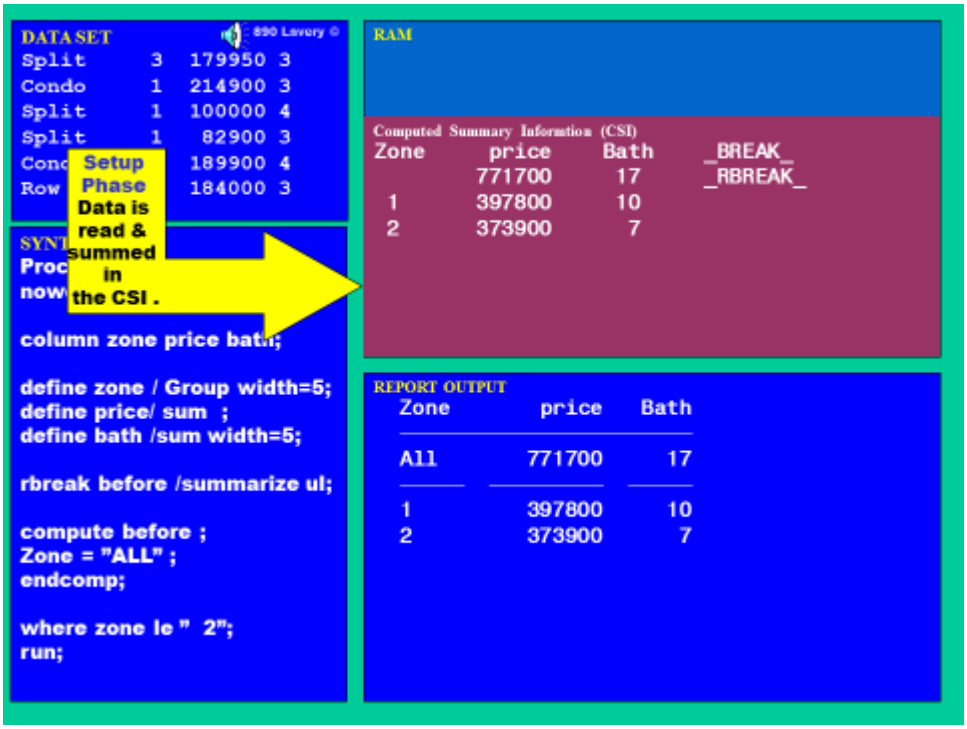

Figure 2

On the right hand side we see a representation of the computer's RAM holding the PROC REPORT internal file (the Computed Summary Information or CSI). In the bottom right-hand corner, we see the output. A slide represents a complete example.

THE THREE MAJOR STEPS INSIDE A PROC REPORT

In figure 2 you can see that part of the RAM holds what looks like a SAS table. The maroon colored table is an internal file for PROC REPORT and is called the computed summary information or CSI.

The major process of producing a PROC REPORT has three steps:

Step one: The evaluation phase: In this step SAS reads the syntax and creates the structure for the CSI (as well as creating internal code that PROC REPORT will run). Of crucial important importance is the column statement. Variables in the column statement will be in the CSI.

Step two: The setup phase: In this step SAS reads the data from the source file into the CSI. I now think of PROC REPORT is a "big file technique" because it is very efficient. The source file is only read once. If we were reporting on "sales by state" and had millions of individual sales records, they would be read once into the CSI which would have 50 rows of data. Any calculations required would be on the 50 rows of data and would be fast. The setup phase really makes PROC REPORT a great tool because, by reading the big file only once, we get the report we want in a short amount of time.

Step three: Report row phase: After reading all the raw data from the source file and summarizing to get to whatever grouping level desired, it is common for a programmer to want to perform a calculation on each, or at least some, of the rows in the CSI. The CSI is processed from top to bottom, one row at a time, and calculations are performed on each row. After all the calculations have been performed, a row is sent to an output data set to be routed through the ODS and displayed in some format.

RULES

The major deliverable of this talk will be showing the reader pictures of the internal process of PROC REPORT. This will allow a reader to "see" what the words in the documentation mean. I want to collect all of rules in this one place, so that this paper can be used as a reference.

I do not expect just reading the rules will offer much enlightenment. It is suggested that you read this paper in three steps. First, read the rules quickly. Second, read the rest of the paper and see how the rules are applied. Third, come back and read the rules again. I only expect that the rules will make any sense if you have had a chance to study some of the examples contained in the rest of the paper.

*The Column statement creates variables in the Computed Summary Information.
To use a variable, from a data set, you request it via the column statement.
* Variables mentioned in the column statement will appear in the CSI
*If you need a variable in a calculation, but do not want to print it, list the variable in the column statement and define it as "NOPRINT".
*Define statements assign characteristics to variables (group, order, across, sum, NOPRINT).
*Rbreak before/after, or compute before/after create a "report level" summary line in the CSI.
*Break before/after var, or compute before/after var, create a "variable level" summary line in the CSI.
*Summary lines in CSI are not printed unless you code a break line with a ***/ summarize*** option.
*Compute variable blocks execute on every line in the CSI. Please note that they execute 1) in the order in which variables appear in the column statement and 2) that all compute var. blocks execute before the Compute Before/Compute After blocks.
*Compute before Var blocks create lines in the CSI and then execute when SAS is processing that line, in the CSI, that is before/after a new value of the variable.
*Compute before blocks (no variable mentioned) create lines in the CSI and execute when SAS is processing that line, in the CSI, that is before/after <u>all</u> other lines in the CSI
* The CSI always contains a _break_ column that is used by PROC REPORT to identify the level of summarization for that row and "trigger" internal processes.
* *Summary lines in CSI are not printed unless there is an associated Break statement with a /summarize option. This can seem a bit confusing but giving programmers control of the process is a good thing and allows the creation of complicated reports.*

EXAMPLE 1: AN ILLUSTRATION OF BASIC PROC REPORT STEPS

Figure 2 shows all the steps "at the same time" and is inaccurate. It's difficult to illustrate a time sequence on a printed page.

Stage 1: In the evaluation stage, the column statement is read and used to create the CSI structure. At this point, the CSI is a table, with the metadata information we might see from a PROC CONTENTS, but with no rows of data.

The CSI will always have an extra column called _break_. This is created by PROC REPORT and is used, by PROC REPORT, to identify rows that are summary rows as opposed to rows containing detailed data. The first row in this CSI is of type _R break_. This tells PROC REPORT that this line, in the CSI, is a *report level summary*. Any numbers in this row will be describing the whole input data set (or at least the rows from the input data set that made it through the where clause – here we coded where zone LE 2 just to make the output fit on a slide).

PROC REPORT is fast and here is one trick that makes it fast. In this example, as each observation is read into the CSI it affects two rows. Each observation will affect the summary row and also affect the role for its zone. Having observations change multiple totals is very powerful and a great speed trick. It is easy to see that the total number of bathrooms where zone is less than or equal to two is, in fact, 17.

At this stage, because zone is defined as “group”, PROC REPORT knows that it will group data by zone. Price and baths are defined as type “sum”. We could ask for other types (Min, Max average and others). By asking for type “sum” we are going to get the totals of the prices and bathrooms, not only on rows for zone one and zone two – but also for the report level row. We will come back to this point later.

In step three, the Report row phase, the CSI is read from top to bottom, one row at a time, and appropriate calculations are performed.

The code has a “compute before” block and SAS knows that a “compute before” statement should only be executed on the row “that is before any other rows in the CSI” and is type _Rbreak_. “Compute before” and “Compute after” statements only execute on rows of the type _R break_ but the individual compute blocks are smart enough to know whether they should execute on the first, or on the last, row in the CSI.

A “Compute before” statement is used to make the output more understandable. It changes the zone, on the first line, to “ALL” and that makes the report easier to read. You should note that zone was defined as character and was also defined as being wide enough to hold the character string “ALL”. If the zone were numeric, or was character, but not wide enough to hold three letters, we would have a problem.

EXAMPLE 2: AN ILLUSTRATION OF TIMEING IN THE REPORT ROW PHASE

To make various “cut and pastes” of output fit on these slides I had to use unusual abbreviations for the variables.

STY stands for style of house. REGN stands for the region of the city in which the house is located. DETL is a “made up” variable that has no logical connection to selling houses. It is a detail variable and, by that I mean, it will show up on every row in the CSI.

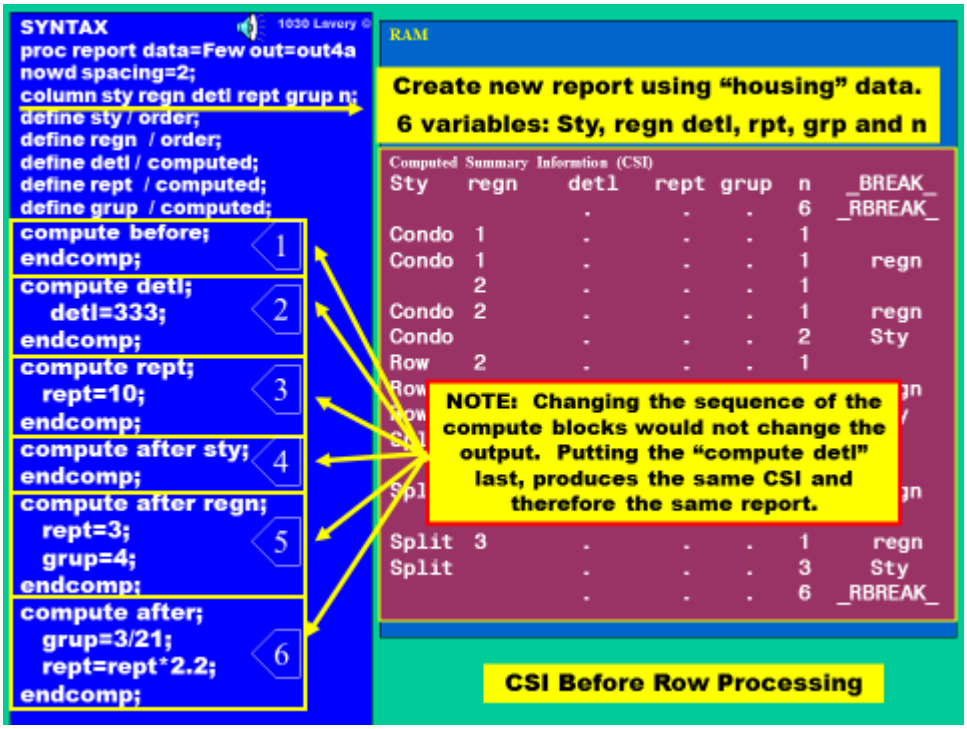

Figure 3

REPT stands for report and this variable has no logical connection to selling houses. It is just a name. GRUP is a variable that has no logical connection to selling houses. The variable name is pronounced like group and I just want to show this variable values to “the group of people attending the seminar”.

Please start by looking at the code on the left-hand side of figure 3. The column statement lists the variables that will be in the CSI. The rightmost variable is n, a PROC REPORT reserved word that causes PROC REPORT to count the number of rows. STY and REGN are “order” variables and DETL, REPT and GRUP are “computed”. In this example, we will compute most of the values in the CSI. This is how the CSI would look at the end of stage II – the setup stage.

As figure 3 says, changing the sequence of the “code blocks” does not affect the order in which blocks execute. Blocks are aligned to variable names and execute in the order in which the variables appear in the column statement.

I’d like to discuss the different compute blocks that were coded in this example. Combined they created a CSI (that we can examine using the out = option) that will allow us to deduce the timing of the execution of compute blocks.

Compute block one creates a row in the CSI but does not cause any calculations to execute. This might seem a bit weird but is a characteristic of a PROC REPORT that a programmer can exploit.

Compute block two is a “compute var” block and it computes a variable in the CSI. It will set the value of detail to 333. Important rules concerning “compute var” blocks are 1) “compute var” blocks execute on every line in the CSI and 2) execute before any “compute before” or “compute after” blocks

Compute block three is a “compute var” block and sets the value of REPT to 10.

Compute after style is a “compute after var” block it executes only on the lines in the CSI that occur after STY changes value. This block does not actually request that any calculations be done. It was intended to show that the “compute before var” and’ compute after var” blocks create rows in the CSI.

We should remember that STY was defined as an order variable and that caused all the rows of a particular style of house to be “bunched” together and also to have the groups sorted.

Block five is also a “compute after var” block. It causes REPT to have a value of 3 and GRUP to have a value of 4. Block five was created to allow a reader to see, on which lines, those kinds of statements execute.

Block six is a “compute after” block. We should remember that, the CSI is processed from the top row to the bottom row and that PROC REPORT performs “appropriate” calculations on each row of the CSI. The statements in block six will only execute on the last row of the CSI – where the value of _break_ is _Rbreak_ and the row is the last row of the CSI.

Figure 4 focuses on the issue of compute blocks and when they execute. If you are going to do complicated calculations in a PROC REPORT you must know the timing of the different compute blocks.

Figure 4 shows the CSI after all the compute blocks have executed and so it can be used to illustrate the rules listed above.

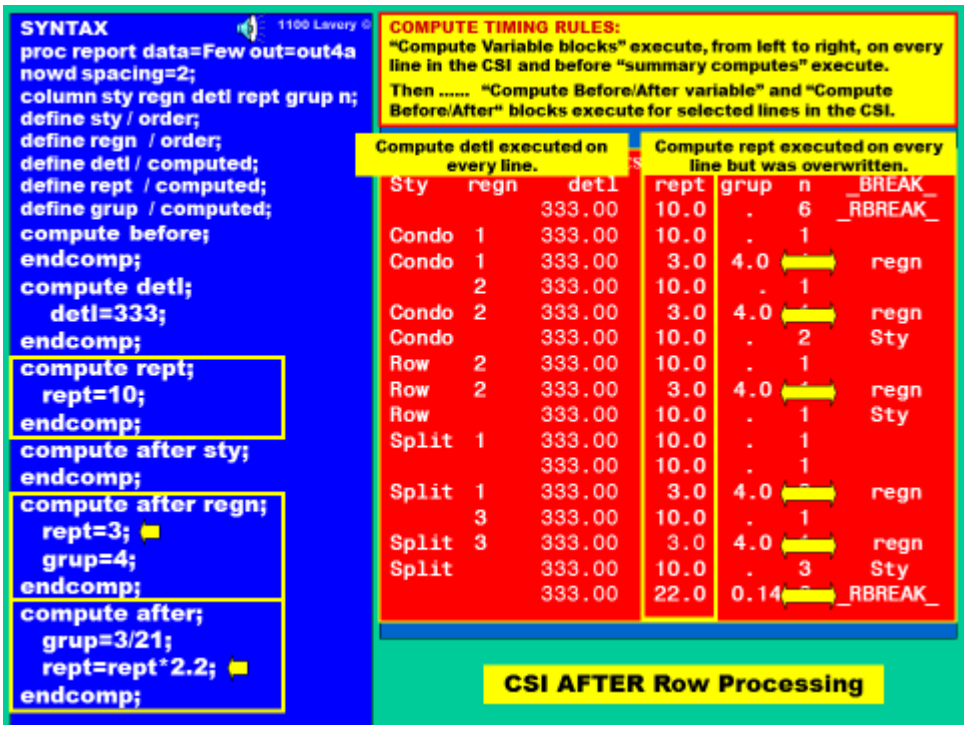

Figure 4

As a start, let's take a look at the column _break_. It has values of _R break_, regn and sty. These values identify rows that contain summary level information. They identify rows where grouping variables have changed values and, with proper coding, these rows in the CSI can occur before, or after, the variable changes value.

If you look at previous figures you can compare the raw data with the value for the n variable that occurs on summary lines under the variable n. A reader can confirm that n, in these rows, is holding summary level data, generally, for the rows above. The first row in the CSI was created by the "compute before" block and contains data (n is six) that summarizes rows that follow.

DETL has a value of 333, even on the first and last rows of the CSI, because "compute var" blocks execute on every line of the CSI – and they execute before any other compute block.

REPT has a value of 10 on most rows, but not all. On some rows the value of REPT is three. One must ask if the values of REPT were, at one time, 10 and then were changed to three or if they were always valued at three. Whenever REPT has a value of three GRUP has a value of four and this suggests that the value of three came from the "compute after regn" block.

The "compute after sty" block just adds a line to the CSI. A programmer will often want to create lines in the CSI that she does not print and we will explore this more in later examples.

The last compute block, the "Compute after" block, gives strong evidence as to the timing of computes. It instructs to take the current value of REPT and multiply it by 2.2. It also overwrites the value of GRUP. If you look in the CSI, REPT has a value of 22. The only way this could occur as if REPT had been set to 10 by a "compute var" block and then multiplied by 2.2. This is strong evidence that "compute var" statements execute on every line and execute before other compute blocks execute.

Understanding the timing of execution of compute blocks is critical to programming complicated reports.

EXAMPLE 3: IF STATEMENTS AND HOW DUMPING THE CSI IS CONFUSING

If statements can be useful when creating a complicated report. Coding if statements is a bit tricky and I use a three step process. Step 1: Before writing any if statements, I write the program without if statements and "dump" the CSI to a file - which I print. It is good to see the CSI because not all rows in the CSI are printed to the SAS listing and I like to see the "missing" rows.

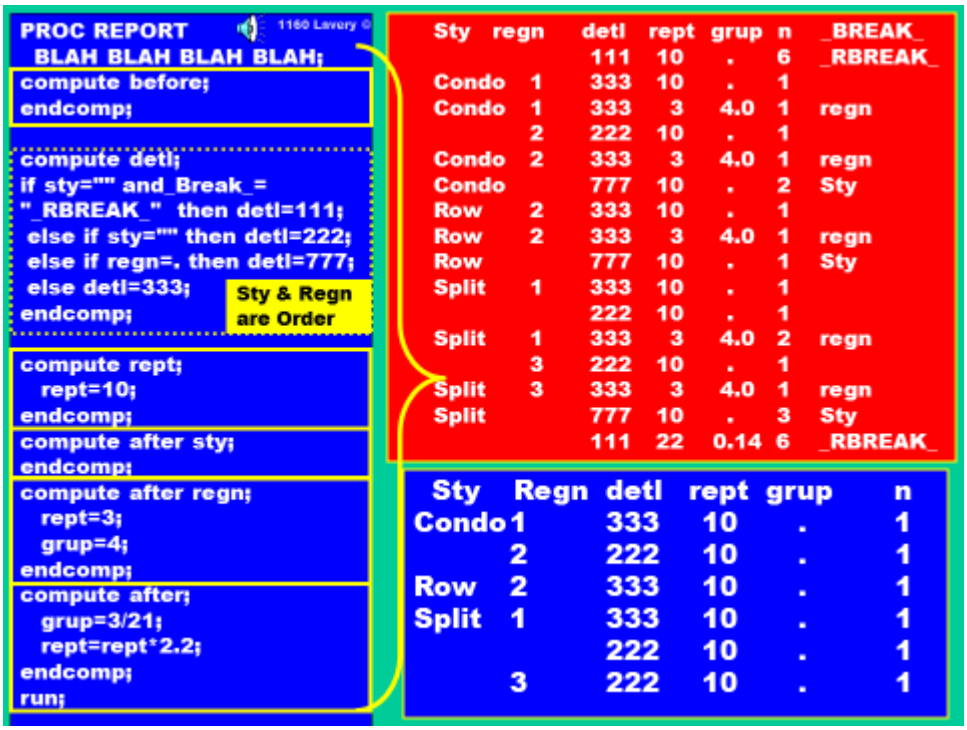

Figure 5

However; there is a problem with "dumping" the CSI. When we use the out = option to "dump" the CSI to a file SAS fills in any missing values that had been caused by values repeating over several lines (please see the STY variable in figure 5). In the CSI, these values are missing and so one cannot code if statements by simply looking at a "dump" of the CSI. If you look in figure 4, the fourth row does not show the word "condo". That value is missing in the CSI and in the listing but not in the "dump" of the CSI.

Step 2: I print the listing/output from the PROC REPORT. This does not show all of the rows in the CSI but does show where repeated values have been set to missing.

Step three: After comparing what I've seen, in the previous two steps, I'm ready to take a try at coding the if statements. If statements require a little bit of thinking because, if one were to get a data refresh with more rows and more repetitions, the pattern of missing values might change. We will examine a few rows in The CSI in figure 5.

The first row came from a "compute before".
The second row is for one house that is a condo in region one.
The third row is a summary row, because we have reached the end of condos in region one. Note that _RBreak_ has the value of regn,

The fourth row, with the missing value for STY, is actually a row describing a condo in region two. "Condo" is suppressed because it would be a repetition of the word "condo" in the row above. Putting underlines on summary lines makes reports a lot more readable.

The fifth row is a summary for condos in region two note that the column _break_ is valued as regn. This is a REGN level break line.

The sixth row is a summary for all the condos. REGN is missing on this row and that makes sense. This row summarizes condos for more than one region. It summarizes condos over region 1 and region 2. It makes sense that REGN is blank on this row. Notice that _break_ has the value of STY on this row.

I also ask you to look at the output at the bottom of this figure. Compute statements cause the creation of rows in the CSI but they do not cause those rows to print. Only break statements with a **/summarize** option cause summary rows in the CSI to print. Because of this, the printed output, shown in the bottom right-hand corner of this figure, is much smaller than the CSI. This looks weird but is a bit of brilliant programming logic.

Programmers have good reason to have rows in the CSI that they do not print. These rows allow a programmer to compute running percentages as we will see in the next example.

EXAMPLE 4: PERCENTAGES WITHIN GROUPS OR "RETAINING" IN PROC REPORT

This example covers a pretty complicated topic and will require several slides. Even the code for this example would not fit on one slide.

Figure 6 only shows part of the code for this example.

This report does not make a very compelling business story. People might say that managers would not want this report, and might be right.

Figure 6

However; this report does fit on a slide and demonstrates several important concepts.

We want to use PROC REPORT to compute the total sales inside a zone. The report (lower right corner) looks useful.

We also want to compute, for each style of house in a zone, the two percentages of sales. We wish to see, for each row, the row's percent of the total dollars in the data set and the percent of the total dollars in the zone.

To calculate these percentages, we need the denominators for the calculations to be stored somewhere in RAM. The denominators will be stored in ram, but separate from the CSI, in what are called "temporary variables". Think of them as the similar to the memory buttons that you have on your calculator. You can store values in temporary variables and recall the values when you want to use them in a calculation.

Figure 7 shows the rest of the code for this example.

AllPr (all zone prices) and ZnTot (Zone Price total) are temporary variables.

Temporary variables cannot be named in the column statement – because variables in the column statement become part of the CSI.
Temporary variables appear in at least two compute blocks.

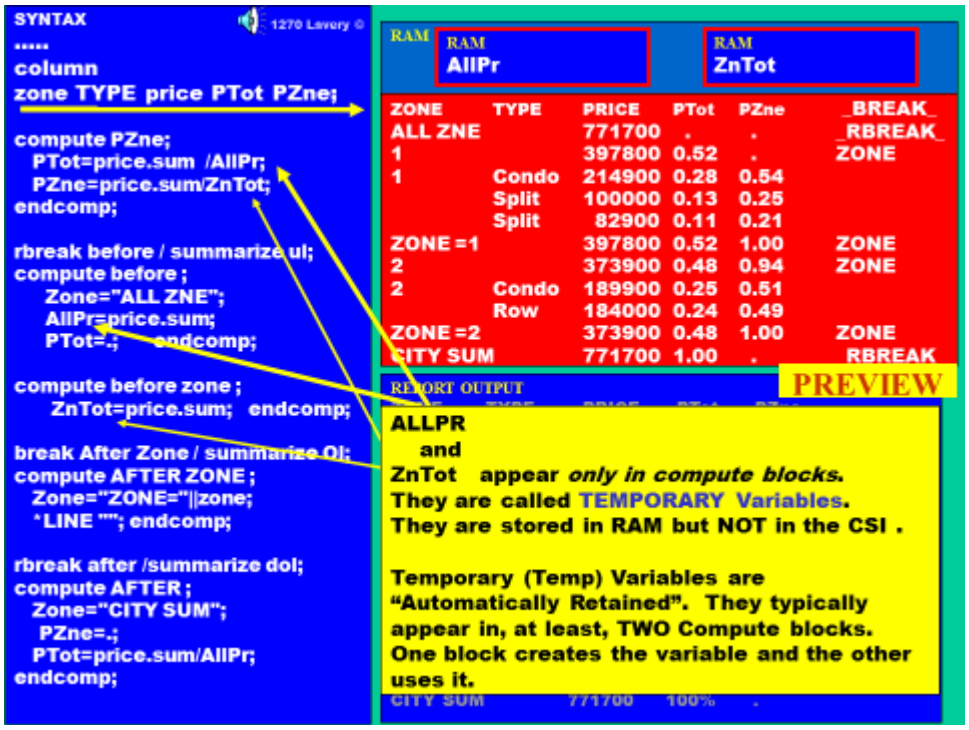

Figure 7

At least one of the compute blocks will move data from the CSI into the temporary variable, as "the proper" row in the CSI is being processed. At least one of the compute blocks will recall values from the temporary variable so that it can be used in a computation. In figure 7, you can see that AllPR and ZnTot do not appear in the column statement and do appear in compute blocks.

In figure 8 we start to process the first row of the CSI. The "rbreak before / summarize ul" will make this row print and underline it in the listing.

In the "Compute before" statement we change the value of zone to be a more useful string.

Figure 8

We take the value of price for this row, which is the sum of the dollars in zones 1 and 2, and **move it from the CSI to the temporary variable.** This is how you store of value in a temporary variable.

I also set Ptot to be missing, though this is not required. Ptote is missing already and I was just being cautious (or confused) when I wrote this code.

"Compute Var" blocks execute on every CSI row. The statements in the "Compute PZne" block execute on this row but return missing because AllPr and ZnTot are missing.

Figure 9 shows us processing a "zone total row". It is not printed because there is no break line with a **/summarize** option.

This code loads the total dollars for zone 1 into ZnTot.

Figure 9

Notice that there are two gold arrows from this compute block. This statement will execute a second time – just as we start processing rows for zone two.

In figure 10, we see an example of processing rows of data inside a zone.

The "Compute PZne" block executes and the two percentages are calculated.

This pattern repeats until the end of the block.

Figure 10

Figure 11 shows the processing of the last row in zone one. We change some text to make the report easier to read.

We show the total dollars for this zone.

The “Compute PZne” block calculates dollars as percentage of the total (.52) and of its own (1.00).

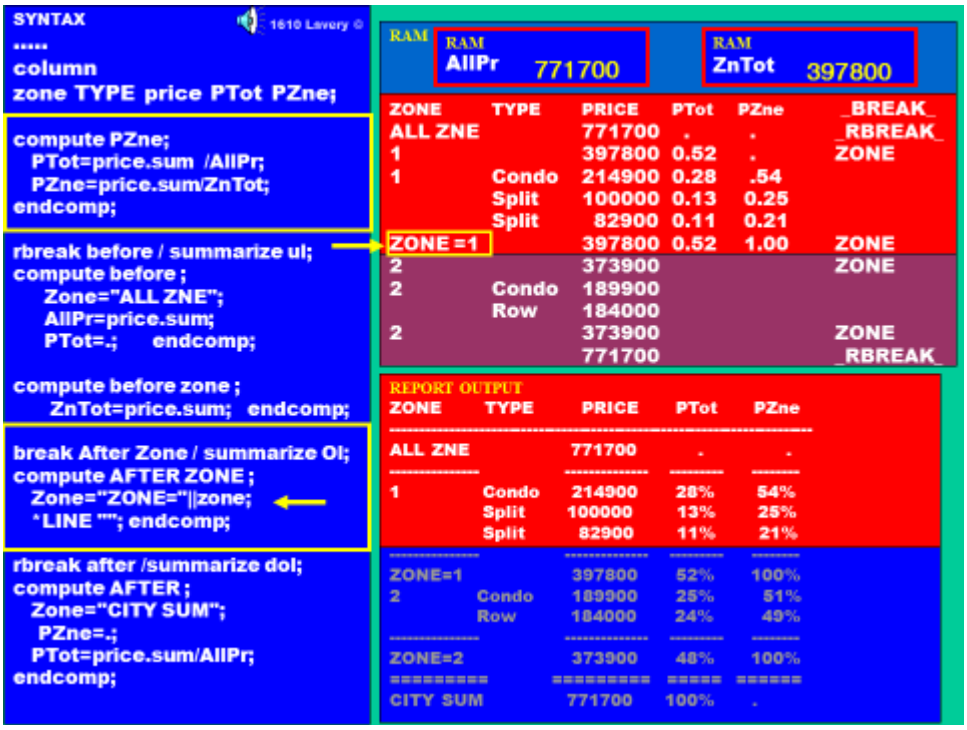

Figure 11

The next row of data will allow us to change the value of ZnTot.

In figure 12 we see the processing for the compute before zone block.

It just moves the value of price (which has been defined as a sum variable) from the CSI to the temporary variable ZnTot.

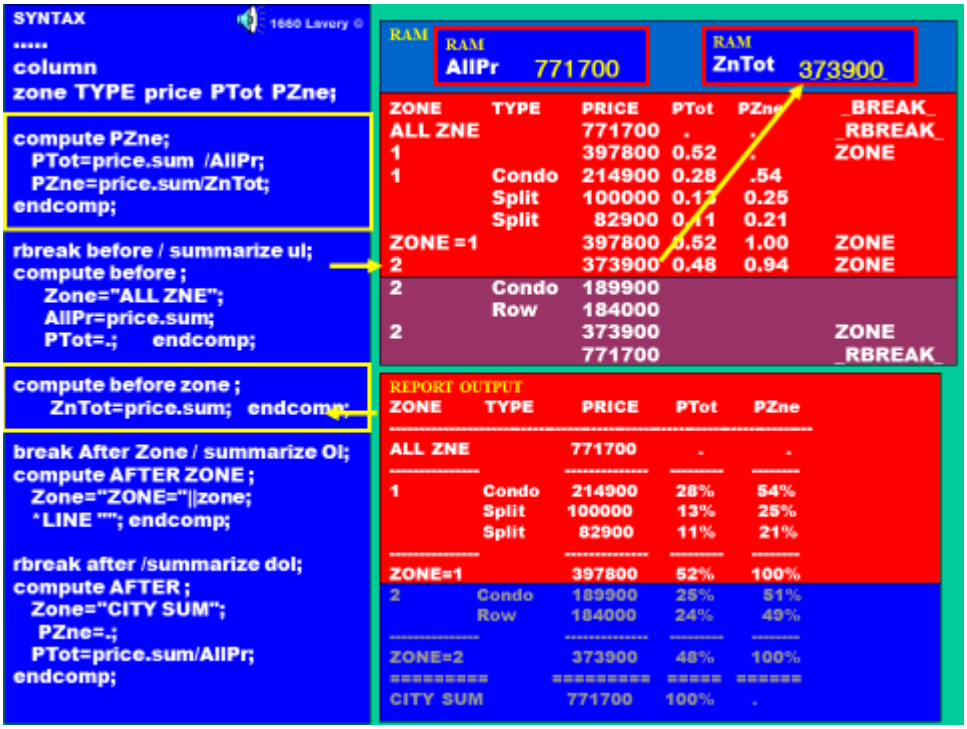

Figure 12

We now have a divisor we can use in calculating percentages for zone two. Please note that we did not have to change the divisor for the Ptot which is stored in AlllPr.

Figure 13 shows the calculations for a typical row inside of zone two.

Calculations are done because the “Compute PZne” block executes ad executes on every row.

A similar pattern is followed until we hit the summary row for zone two.

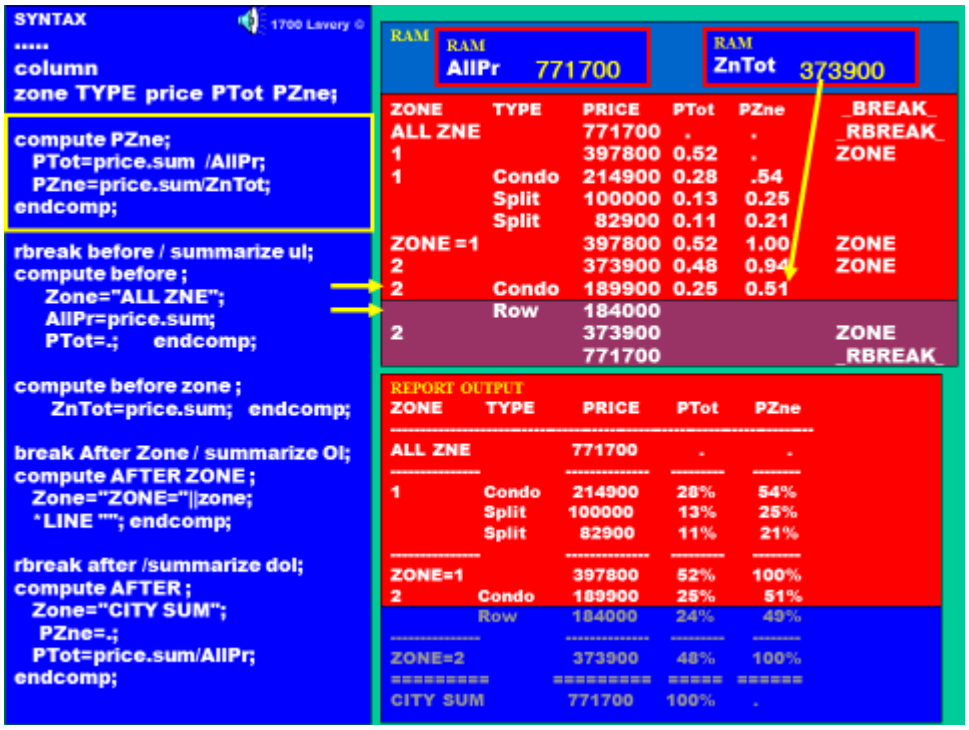

Figure 13

Figure 14 shows the processing for the summary row for zone two.

Two blocks of code execute.

They are:
”Compute PZne” and
“Compute After Zone”.

The “break after Zone / summarize;” will cause the line to print.

Figure 14

Figure 15 shows the processing for the final row in the CSI.
To make this final row of output, **two sections of code must execute**.

First the "Compute PZne" block executes to calculate the percentages.
Pzne is calculated and is calculated wrong. SAS divides 771700 by 373900. SAS resets this to missing in the "Compute after block".

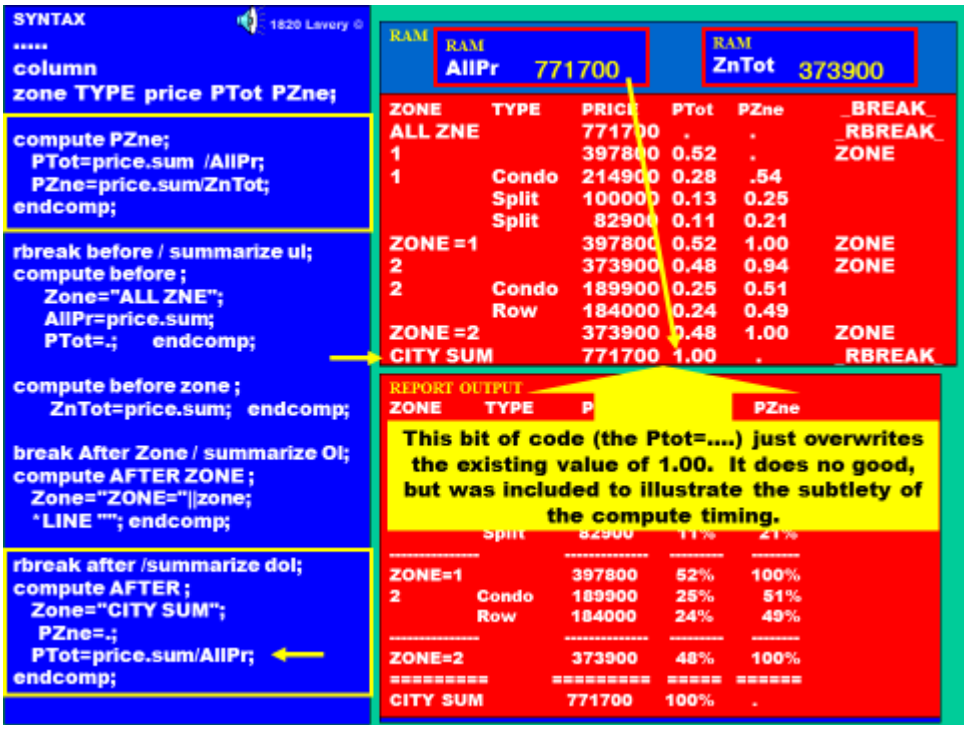

Figure 15

Then the "Compute after" block of code executes and this block deserves a bit of discussion.
We change the text in the variable "zone" to "CITY SUM" to make the report easier to read.
PZne is set to missing. PZne was calculated incorrectly and, besides, does not make logical sense on this row. We calculate PTot again, but this was not required. PTot had been calculated correctly in the "Compute PZne" block

REFERENCES

It is useful to know the timing of calculations when coding complex PROC REPORTs.This talk – the PowerPoints and my voice, in its full 2 ½ hour length- was burned onto a CD and included in the back of the hardcover (only) version of Art Carpenter's excellent book on PROC REPORT. If you have a chance to buy this book I would do so – especially if you can get the hard cover version. I appreciate the chance I had to work with Art on this project.

ACKNOWLEDGMENTS

Thanks to all the helpful folks at SAS and especially those at Tech Support.

RECOMMENDED READING

Carpenter's Complete Guide to the SAS REPORT Procedure (Sas Press)

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:
Russ Lavery Russ.lavery@verizon.net